УДК 004.89

ЭФФЕКТИВНЫЙ АЛГОРИТМ СИНТЕЗА ПРОГРАММ С УСЛОВИЯМИ И ПОДПРОГРАММАМИ

А.Е. Пинжин, В.Б. Новосельцев

Томский политехнический университет
E-mail: alex_pinjin@tpu.ru, Vitalii_Novoseltsev@tpu.ru

Предлагается алгоритм синтеза программ с условиями и подпрограммами на основе заданной спецификации. Алгоритм позволяет добиться высокой производительности за счет предварительной подготовки специальных структур данных. Затраты на вывод и извлечение программы характеризуются линейной функцией от функциональных связей, объявленных в спецификации. Приведены результаты опытного сравнения с существующими алгоритмами.

Ключевые слова:
Функциональная связь, алгоритмы, логический вывод, синтез программ, функциональное программирование, подпрограммы, условия.

В [1] изложено описание и реализация алгоритма синтеза линейных программ, обладающего линейными показателями скорости вывода относительно количества атрибутов и функциональных связей в исходной схеме. В настоящей работе отражено расширение алгоритма для решения задач синтеза ветвящихся программ с условиями и программ с подпрограммами с сохранением линейной эффективности.

В [2] для осуществления синтеза программ с подпрограммами перед началом планирования предлагается осуществлять развертку подсхем. В [3, 4] излагаются возможные способы вывода без предварительной развертки, однако эти результаты требуют более детального исследования и подтверждения экспериментальными результатами. В настоящей статье предлагается описание и реализация стратегии вывода на динамически генерируемой и управляемой выводом развертке схемы. Такой алгоритм является более эффективным, т. к. не требует затрат на дополнительную подготовку структур данных, использует меньший объем внешней памяти и не затрагивает подсхемы, не участвующие в планировании на конкретной задаче.

Базовые определения и постановка задачи

Исходные данные для процедуры планирования (синтеза) *линейных* программ поставляются в виде множества имен атрибутов и функциональных связей.

Функциональная связь (ФС) определяется выражением вида f: $a_1,...,a_n \to a_0$, где f – имя, a_i – аргументы, a_0 – результат ФС. Совокупность атрибутов и ФС объединяются в схему вида $T=(a_0,a_1,...,a_n \mid f\_set)$, где T – имя схемы, $a_0,a_1,...,a_n$ – список атрибутов схемы, f_set – множество ФС схемы.

При постановке задачи синтеза ветвящейся программы в описание схемы вводится *альтернативная* или *вариантная* часть схемы, определяемая записью вида *if* $p_1(...) \supset a_{10}$, $a_{11}... \mid f\_set_1 \square ... \square$ $p_k(...) \supset a_{k0}$, $a_{k1}...$ *endif* $\mid f\_set_k$.

Части условия, разделенные символом $\square$, будем называть *ветвями условия*. Символ $\square$ соответствует традиционной программной конструкции *else-if* или ветви оператора *case*.

$p_1,...,p_k$ – выбирающие селекторы – представляют собой логическую функцию вида $p_n(a_{n0},...)$, где $a_{n0},...$ – атрибуты заголовка схемы, определяющую *допустимость* условных атрибутов и ФС, входящих в условную ветвь.

Следом за описанием селектора указываются *условные атрибуты*, имеющие смысл лишь в ветви условия, в заголовке которой они описаны.

Выражение *f_set*, следующее за описанием условных атрибутов, скрывает множество *условных ФС*, которые имеют смысл только в соответствующей ветви условия. Условная ФС может включать в качестве аргументов и результатов атрибуты заголовка схемы и условные атрибуты своей ветви условия.

Факт наличия условия p у некоторого атрибута a или функциональной связи f будем обозначать выражениями вида a/p и f/p соответственно.

Отметим, что в настоящей работе рассматривается реализация программ с двухальтернативными условиями без вложенных условий. Эти ограничения, вероятно, будут устранены в будущем, а на текущий момент могут быть преодолены с помощью подпрограмм. При двухальтернативной вариантной части условия, соответствующие ветвям, могут быть обозначены через p и $\sim p$.

Для того чтобы дополнить описываемую теорию понятием *подпрограммы* (подпрограммы, процедуры, программного метода) необходимо ввести понятие *структурной вычислительной модели* (С-модель) [2–4], которая представляет собой конечную совокупность схем $M=(T_1,...,T_s)$.

Теперь введем понятие *тип атрибута*. Под типом атрибута понимается принадлежность возможных значений атрибута определенному поименованному домену. Выделим *элементарные* или *первичные* типы (например, целое число, вещественное число, строка) и *сложные* или *непервичные* типы, соответствующие описанию схемы, входящей в С-модель. Для целей синтеза программ с подпрограммами нас интересуют в первую очередь

второй подкласс типов. Будем обозначать принадлежность атрибута t типу T выражением вида $t{:}T$, где t – имя атрибута, T – имя схемы. Для упрощения записи указатель принадлежности атрибута элементарному типу может быть опущен. Внутренние атрибуты схемы T, принадлежащие атрибуту $t{:}T$ выражаются записью $t.a_0, ..., t.a_n$. Отметим обязательное ограничение для С-моделей – каждый тип атрибута, встречающийся в определениях схем С-модели M, должен являться элементарным, либо соответствовать схеме, принадлежащей M.

В предлагаемом подходе в процессе планирования каждому атрибуту непервичного типа ставится в соответствие подпрограмма. В терминах классических языков программирования запись внутреннего атрибута $t.a$ соответствует фактическому параметру процедуры, а объявление атрибута a в заголовке схемы T является ее формальным параметром.

Постановка задачи синтеза может быть переформулирована относительно [1] следующим образом: на основе С-модели M требуется сформировать процедуру $S=(t{:}T,A_0,X_0,)$, где t – непервичный атрибут схемы $T\in M$, на которой ставится задача, A_0 и X_0 – наборы имён соответственно исходных и искомых атрибутов, содержащихся в T. Задача синтеза состоит в генерации подпрограммы на основе модели M, принимающей на вход набор A_0 и возвращающей X_0.

Для пояснения введенных обозначений приведем пример модели, предоставляющей возможность решения простейшей системы уравнений:

$$y=\begin{cases} x; x<0 \\ x+n; x>=0 \end{cases}, \text{ где } n=x^2.$$

Основная схема:

$T = (x, y$

if $p_1(x)\supset | f_{11}: x\rightarrow y \square ... \square p_2(x)\supset r{:}R \,|\, f_{21}: x\rightarrow r.m;$

$f_{22}: r.n, x\rightarrow y$ *endif* $)$

Схема для вычисления квадрата числа:

$R = (m, n \,|\, f_{sq}: m > n)$.

Трансформация вариантной части

При выполнении планирования на схеме, содержащей вариантную часть, необходимо обеспечить доказательства достижимости атрибутов, входящих в логическую функцию селектора. В настоящем подходе перед началом планирования для каждой ветви условия предлагается искусственно ввести некий *виртуальный атрибут*, являющийся результатом логической функции селектора:

$$p_n^{virt}=p_n(a_{n0},...),$$

где p_n^{virt} – виртуальный атрибут; p_n – селектор.

Далее, функция селектора вводится в набор *f_set* ФС основной части схемы, а виртуальный атрибут вводится в число атрибутов заголовка схемы и добавляется в качестве обязательного аргумента функций, доставляющих атрибуты в соответствующую условную ветвь. Таковыми являются условные ФС, аргументами которых являются безусловные атрибуты. После проведенных преобразований, описание схемы может быть представлено следующим образом:

$T = (a_0{:}S_0,...,a_n{:}S_1, p_1^{virt}, p_2^{virt}$

if $p_1^{virt}\supset a_{10},...| f_{11}{:}a_i,..., p_1^{virt}\rightarrow ..., ... \square ... \square p_k^{virt}\supset a_{k0},...|$

$f_{k1}: a_j,..., p_k^{virt}\rightarrow ..., ...$ *endif*$|$

f_set, $p_1^{virt} = p_1(...), p_k^{virt} = p_k(...))$.

Подготовка структур данных (компиляция)

Перед началом планирования выполняется специальная подготовка – компиляция схем С-модели. Заметим, что результат компиляции не зависит от исходных и целевых атрибутов и может быть использован многократно для выполнения разных задач планирования на модели. Эффективная подготовка исходных данных является отдельной задачей и не учитывается при оценке эффективности.

Исходными данными для процедуры поиска доказательства являются (рис. 1):

- набор объектов типа `Схема`;
- пара объектов `ВетвьУсловия`, если схема обладает вариантной частью. Каждый объект содержит указатель на соответствующее условие, а также изначально пустой список шагов доказательства, принадлежащих этой ветви. Каждый объект `Условие` однозначно соответствует ветви условия и содержит имя условия и признак отрицания;
- список объектов типа `ФункциональнаяСвязь`. Содержит счетчик достижимых аргументов, который используется при выводе и ссылку на атрибут – результат ФС. Если на ФС наложено некоторое условие (*f/p*), то при компиляции определяется ссылка на соответствующий объект типа `Условие`;
- список объектов типа `Атрибут`, содержащий непервичные атрибуты вида «*t*:*T*», определяемые ссылкой `сложныйТип`; субатрибуты непервичных атрибутов вида «*t.a*», которые являются доставляющими атрибутами вызовов подпрограмм (фактические параметры) и обладают ссылкой `вызовПодпрограммыАргумент` и рекурсивной ссылкой `субАтрибут` на соответствующий атрибут вызываемой подсхемы (формальный параметр); элементарные атрибуты текущей схемы, которые могут обладать ссылкой `фсАргумент` на ФС, где они участвуют в качестве аргумента.
- если на атрибут наложено некоторое условие (*a/p*), то при компиляции определяется ссылка на соответствующий объект `Условие`. Список `условияДопустимости` изначально также содержит это условие и пополняется в процессе вывода достигнутыми условиями допустимости;

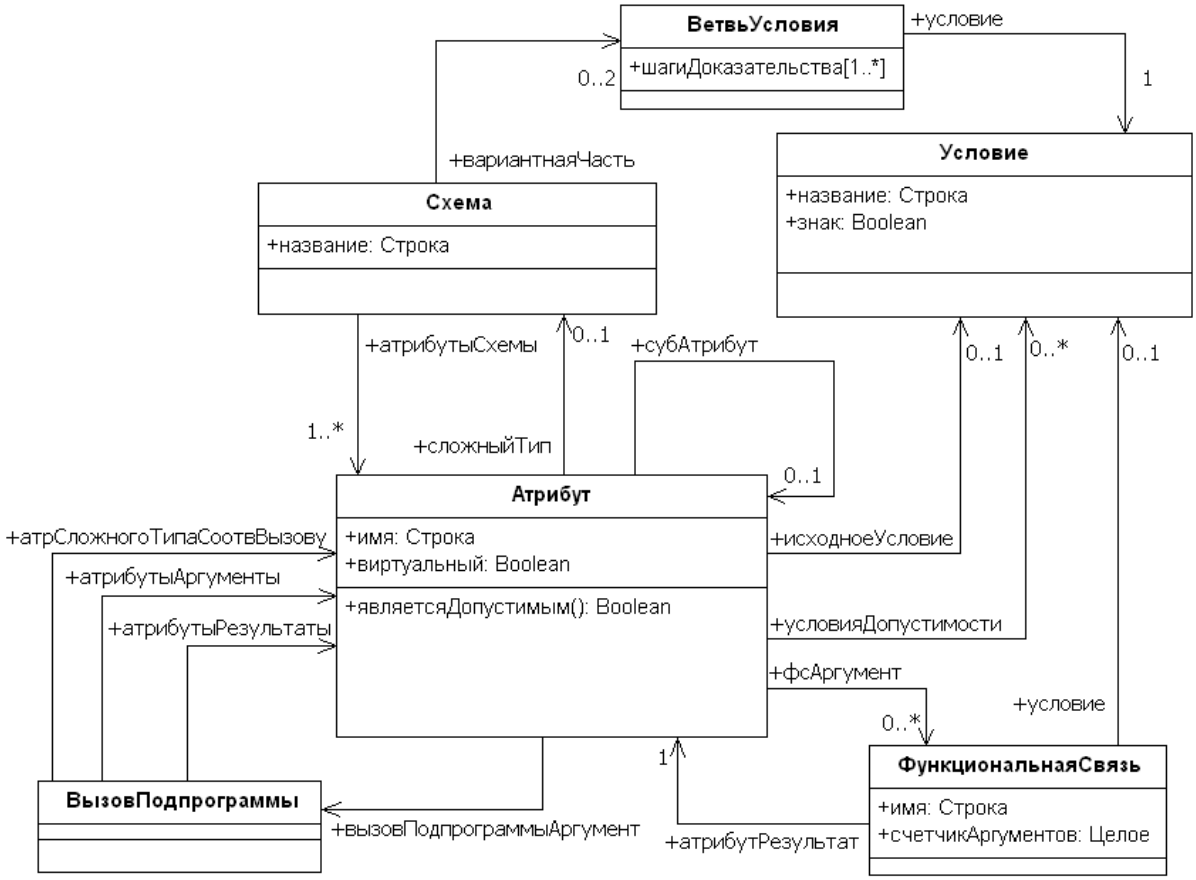

Рис. 1. *Модель исходных данных для алгоритма вывода*

- по каждому атрибуту непервичного типа создается объект типа `ВызовПодпрограммы`. Для каждого вызова объявляется список аргументов и результатов. Аргументы вызова процедуры определяются в ходе выполнения доказательства, атрибуты-результаты – в процессе подготовки структур данных по субатрибутам подсхем, служащих аргументами ФС текущей схемы. В процессе доказательства могут быть достигнуты не все атрибуты-результаты, поэтому их список может быть сокращен.

В процессе компиляции осуществляется и трансформация условий, описанная в предыдущем разделе. Виртуальные атрибуты, созданные в процессе подготовки вариантной части, помечаются свойством `Виртуальный`, которое может использоваться при синтезе программы из доказательства.

Постановка задачи

Для постановки задачи поиска доказательства необходимо создать атрибут непервичного типа на исходной схеме и объявить объект типа `ВызовПодпрограммы` с ссылками на исходные и целевые атрибуты (списки `атрибутыАргументы` и `аргументыРезультаты`). Результатом вывода является список достижимых атрибутов из числа целевых. Сравнивая эти списки можно сделать вывод об успешном доказательстве теоремы. В результате планирования исходный вызов подпрограммы связывается с объектом типа `Подпрограмма` (рис. 2), содержащим шаги доказательства.

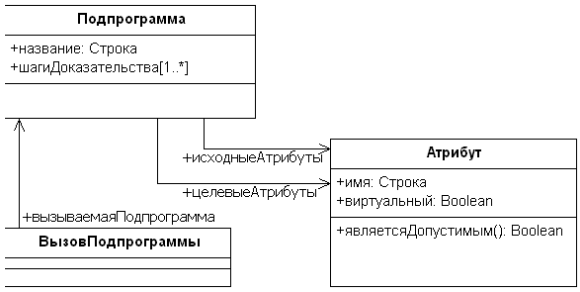

Рис. 2. *Модель подпрограммы*

Шаги доказательства формируются в процессе вывода и представляют собой последовательность объектов, каждый из которых содержит ссылки на достижимые атрибуты и элемент программы, применение которого обеспечивает их достижимость. Выделяются три вида таких элементов (или *программных термов*):

- Функциональная связь.
- Вызов подпрограммы (в свою очередь ссылающийся на подпрограмму).
- Вариантная часть (в виде пары ветвей условия).

Структуры данных для хранения шагов доказательства представлены на рис. 3. Отметим, что

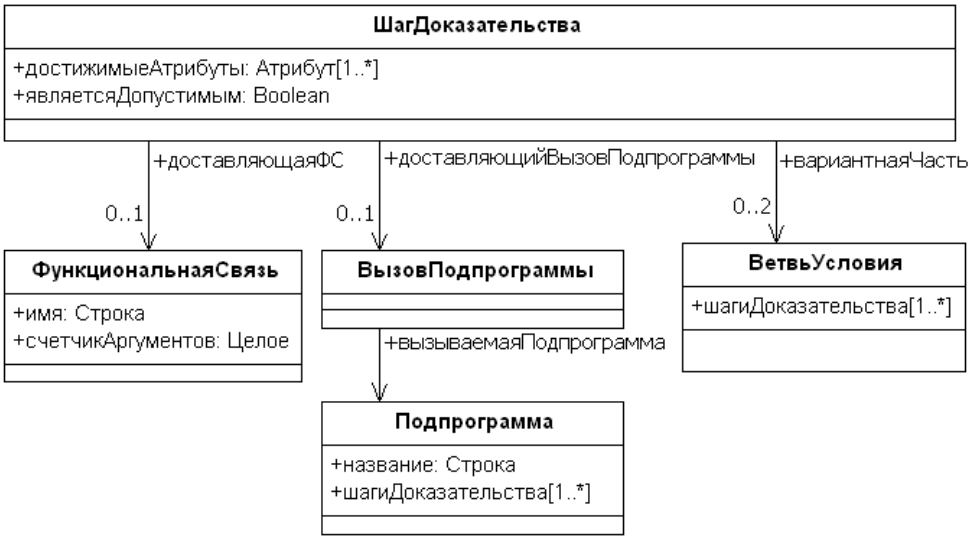

Рис. 3. *Модель результатов вывода*

вполне применимая в представленной модели связь наследования не используется в целях увеличения производительности алгоритма.

Функциональные связи и вызовы подпрограмм являются программными термами, вычисляющие по строго определенному набору аргументов значения определенных целевых атрибутов. Будем называть такие программные термы *предложениями вычислимости* (ПВ).

Алгоритм доказательства

Используя описанные выше входные структуры данных и постановку задачи в виде вызова подпрограммы, алгоритм пытается построить доказательство теоремы существования решения. При этом используется подход со счетчиками аргументов ФС, описанный подробно в [1]. Если при обработке очередного атрибута счетчик аргументов ФС равен нулю (т. е. достижим атрибут – результат ФС), то формируется очередной шаг доказательства. Если на ФС наложено условие, то шаг помещается как в список шагов доказательства текущей подпрограммы, так и в список шагов соответствующей ветви условия. Условие ФС добавляется в список условий допустимости достижимого атрибута (поле `условияДопустимости` объекта `Атрибут`). Затем с помощью метода `являетсяДопустимым`, проверяется, достижим ли атрибут при всех альтернативных частях условия, т. е. при $/p$ и $/\sim p$. Если получен утвердительный ответ, атрибут помещается в список `достижимыеАтрибуты` шага доказательства, содержащего вариантную часть (ветви условия). Этот специфичный шаг доказательства создается заблаговременно для каждой подпрограммы, содержащей вариантную часть, и добавляется в список шагов доказательства подпрограммы однократно при достижении первого безусловного атрибута – результата условных ФС.

Значение, полученное в результате вызова метода `являетсяДопустимым` достижимого атрибута, назначается свойству `являетсяДопустимым` шага доказательства. Если при обработке очередного аргумента ФС выясняется, что ФС является безусловной (т. е. допустимой), а шаг доказательства, в котором был получен атрибут-аргумент, не является допустимым, то уменьшение счетчика аргументов ФС не выполняется. Игнорируется также шаг доказательства, содержащий ссылку на вариантную часть. Он используется в дальнейшем при синтезе программы.

В отношении подсхем при осуществлении вывода используется следующая тактика: в процессе вывода фиксируется текущая подпрограмма, перед началом планирования текущей объявляется процедура, на которой была поставлена задача. Выполняется вывод на ФС текущей подпрограммы, без спуска во вложенные вызовы подпрограмм, которые помещаются во временный стек. Затем, при невозможности дальнейшего планирования, осуществляется вывод на очередной подпрограмме из стека, после чего продолжается вывод на текущей подпрограмме. Процесс поиска решения является рекурсивным – вывод всех возможных аргументов вызова внутренних подпрограмм в текущей подпрограмме, рекурсивный запуск планирования на очередной внутренней подпрограмме, получение списка достигнутых целей и продолжение работы во внешней подпрограмме. При этом повторный вход в подпрограмму не выполняется.

Каждый спуск в подпрограмму приводит к созданию нового объекта типа `Подпрограмма` (изначально присутствуют только вызовы подпрограмм). При спуске и подъеме из подпрограммы с помощью рекурсивной ссылки `субАтрибут` осуществляется перевод фактических параметров в формальные.

Алгоритм вывода можно представить с помощью следующего псевдокода:

```
begin
Создать подпрограмму, зафиксировать ее как текущую
Определить схему, соответствующую подпрограмме
Получить формальные атрибуты, соответствующие аргументам и результатам вызова подпрограммы
По каждому аргументу создать шаг доказательства и поместить в список шагов доказательства текущей подпрограммы
Установить счетчик необработанных шагов равным количеству аргументов
Установить счетчик недостижимых целевых атрибутов равным количеству атрибутов-результатов
while (счетчик необработанных шагов доказательства больше 0) begin
  Получить очередной шаг доказательства из списка
  Получить очередной достижимый атрибут из шага доказательства
  if (достижимый атрибут входит в список целевых атрибутов and
  шаг доказательства является допустимым) then
    Поместить атрибут в список достигнутых целей
    Уменьшить счетчик недостижимых целевых атрибутов на 1
    if (счетчик недостижимых целевых атрибутов равен 0) then
      Теорема доказана, выход из программы
    end_if
  end_if
  Получить список ФС, в которых достижимый атрибут участвует в качестве аргумента
  while (обработаны не все ФС) begin
    Получить очередную ФС
    if (ФС является безусловной and
       шаг доказательства не является допустимым) then
      пропустить обработку ФС
    end_if
    Уменьшить счетчик аргументов ФС на 1
    if (счетчик аргументов равен 0) then
      Получить атрибут-результат ФС
      Создать шаг доказательства и поместить в список шагов текущей подпрограммы
        if (ФС является условной) then
          Получить соответствующую ветвь условия из вариантной части и поместить в нее шаг доказательства
          Добавить условие допустимости в атрибут-результат
          if (атрибут-результат является допустимым) then
            Установить свойство являетсяДопустимым шага доказательства в true
            Добавить атрибут-результат в список достижимых атрибутов вариантной части
            Добавить в список шагов доказательства текущей подпрограммы шаг доказательства - вариантную часть, если не была добавлена ранее
          end_if
        end_if
        Увеличить счетчик необработанных шагов на 1
      end_if
    end_while
    if (достижимый атрибут является аргументом вызова подпрограммы) then
       Добавить атрибут в аргументы вызова подпрограммы
       Поместить вызов подпрограммы в стек
    end_if
    if (счетчик необработанных шагов равен 0 and
          стек вызовов подпрограмм не пустой) then
       Получить очередной вызов подпрограммы из стека
       Рекурсивно запустить процедуру вывода
       Получить достижимые атрибуты-результаты вызова подпрограммы
       Создать шаг доказательства и поместить его в список шагов текущей подпрограммы
       Увеличить счетчик необработанных шагов на 1
    end_if
    Уменьшить счетчик необработанных шагов на 1
end_while
Теорема не доказана, выход из программы
end
```

Синтез программы

Синтез программы осуществляется на основе списка шагов доказательства, полученного в результате планирования. Процесс осуществляется в два этапа: очистка и сборка программы. Очистка включает в себя обработку списка шагов доказательства подпрограммы «снизу-вверх» и удаление элементов, не участвующих в получении целей. Кроме того, из списка шагов доказательства удаляются условные шаги, дублируемые в списке ветвей условия. Процесс очистки является рекурсивным в том смысле, что он выполняется для всех вызовов подпрограмм, встречаемых среди шагов доказательства текущей подпрограммы.

На вход процедуры очистки подается главная подпрограмма. В соответствии со структурой объекта-подпрограммы (рис. 2) из нее извлекаются аргументы, цели и список шагов доказательства, которые служат исходными данными для алгоритма очистки. Рассмотрим этот алгоритм более подробно:

```
begin
  Получить список аргументов
  Получить список целей
  Получить список шагов доказательства
  while (обработаны не все шаги доказательства) begin
    Получить очередной шаг, начиная с конца списка
    while (обработаны не все достижимые атрибуты шага док-ва) begin
      Получить очередной достижимый атрибут шага док-ва
      if (достижимый атрибут содержится в списке целей) then
        if (доставляющее ПВ - ФС или вызов п/п - является условным) then
          Удалить шаг доказательства
        else_if (шаг док-ва явл. вариантной частью) then
          Удалить достижимый атрибут из списка целей
          Запустить процедуру очистки вариантной части
          Получить список входных атрибутов вариантной части
          Добавить входные атрибуты вариантной части в список целей
        else
          Удалить достижимый атрибут из списка целей
```

```
        Добавить аргументы доставляющего ПВ в список целей
        If (доставляющим ПВ является вызов п/п) then
          Рекурсивно запустить процедуру очистки на вызываемой п/п
        end_if
      end_if
    end_if
  end_while
end_while
end
```

Процедура очистки вариантной части в целом повторяет приведенный алгоритм. Основное отличие состоит в том, что очистка выполняется итеративно, по всем ветвям условия. Обработка каждой ветви начинается с базового списка целевых атрибутов, полученных перед достижением шага доказательства, содержащего вариантную часть. По аналогии с приведенным алгоритмом, аргументы условных ПВ, участвующих в достижении целей, добавляются в общий список, а обработанные достижимые атрибуты – удаляются. По окончании работы алгоритма этот список содержит безусловные входные аргументы вариантной части, которые при выходе из процедуры обработки вариантной части добавляются в список целей основной процедуры.

Набор шагов доказательства, полученный в результате очистки, по сути дела является абстрактной схемой программы, реализованной в виде структур, представленных на рис. 3. Процесс сборки зависит от конечных потребностей пользователя и может включать в себя, например, составление текста программы на определенном языке программирования. Так как этот процесс достаточно очевиден и в то же время весьма субъективен, авторы не считают необходимым останавливаться на нем подробно.

Оценка эффективности алгоритма вывода

Прежде всего, отметим, что приведенные в этом разделе расчеты позволяют получить лишь приблизительные значения времени работы алгоритма, так как учитываются далеко не все типы временных затрат. Основной целью здесь является математическое обоснование линейных показателей скорости работы алгоритма относительно объема исходной спецификации. Операции, затраты на которые были взяты в расчет при оценке эффективности, содержат следующие действия:

- создание/изменение сложных объектов (например, шагов доказательства);
- помещение или получение очередного объекта из списка.

Игнорируются следующие типы операций:

- сравнения примитивных значений и проверки истинностного значения логических переменных;
- установки значений примитивных переменных.

Также игнорируется операция проверки достижимости конечных целей на каждом шаге доказательства, т. к. она является опциональной и может быть исключена без ущерба для функциональных возможностей алгоритма.

Пусть s обозначает общее количество ФС схемы, а a_j – количество аргументов j-й ФС. Тогда, для случая линейной программы без условий и подпрограмм время работы алгоритма вывода определяется формулой:

$$T = T^{нач} + \sum_{j=1}^{s} (a_j t^{арг\ ФС} + t^{ш\ /д}),$$

где $T^{нач}$ – временные затраты на начальный этап обработки – создание подпрограммы, установка счетчиков, перевод фактических параметров в формальные и др. При достаточно больших объемах исходных схем эти затраты являются пренебрежительно малыми; $t^{арг.ФС}$ – время обработки аргумента ФС – включает в себя время получения очередной ФС, в которую атрибут входит в качестве аргумента, и уменьшение счетчика аргументов ФС; $t^{ш/д}$ – время обработки шага доказательства – включает в себя затраты на его создание, помещение в список и получение очередного необработанного шага.

При обработке схемы с условиями в расчет затрат на доказательство включаются: $t^{пров.усл.ФС}$ – время проверки наличия условия у ФС; $t^{пров.доп}$ – время проверки допустимости ФС (вызов метода `являетсяДопустимым` у достижимого атрибута); $t^{поиск\ усл.ветви}$ – затраты на получение ветви условия, соответствующей условию ФС.

Для предельного случая, когда все ФС являются условными:

$$T = T^{нач} + \sum_{j=1}^{s} \left(\begin{array}{l} a_j t^{арг.ФС} + t^{ш\ /д} + t^{пров\ усл\ ФС} + \\ + t^{пров.доп} + t^{поиск\ \_усл\ ветви} \end{array} \right). \qquad (*)$$

При осуществлении вывода на схеме с подпрограммами для каждой доставляющей ФС в расчет затрат включаются затраты $t^{арг.\ п/п}$ – на добавление аргумента вызова подпрограммы; $t^{стек\ п/п}$ – на добавление/получение вызова подпрограммы из стека.

Пусть также r обозначает количество схем в исходной спецификации, u_i – количество непервичных атрибутов (т. е. количество вызовов подпрограмм) в i-й схеме, а b_{ik} – количество ФС, доставляющих вложенные атрибуты k-го непервичного атрибута в i-й схеме (т. е. количество аргументов вызовов подпрограммы).

Таким образом, для предельного случая, когда в результатах планирования участвуют все подсхемы, общее время работы алгоритма определяется следующей формулой:

$$T = \sum_{i=1}^{r} \left[\begin{array}{l} T_j^{нач} + \sum_{j=1}^{s_i} \left(\begin{array}{l} a_{ij} t^{арг.ФС} + t^{ш\ /д} + t^{пр\ усл\ ФС} + \\ + t^{пров.доп} + t^{поискусл\ ветви} \end{array} \right) + \\ \sum_{k=1}^{u_i} (b_{ik} t^{арг.п\ /п} + t^{стекп\ /п}) \end{array} \right].$$

Примем допущения, что количество:

- ФС каждой подсхемы равно s;

- аргументов каждой ФС равно a;
- непервичных атрибутов в каждой схеме равно u;
- доставляющих ФС каждой подсхемы равно b;

В результате получим упрощенную формулу:

$$T=r(T^{нач}+s(at^{арг.ФС}+t^{ш/д}+t^{пр.усл.ФС}+t^{пров.доп}+t^{поискусл.ветви})+ +u(bt^{арг.п/п}+t^{стекп/п})).$$

Как видно из приведенных расчетов, в случае вывода на схемах с условиями и подсхемами, время работы алгоритма сохраняет линейную зависимость от общего количества ПВ и аргументов этих ПВ, объявленных в исходной спецификации.

Экспериментальная оценка эффективности алгоритма

Тестовые оценки реализации алгоритма вывода осуществлялись на трех типах задач: линейные программы, программы с условиями и программы с условиями и подпрограммами. В качестве сравнительной оценки с существующими алгоритмами приводится результат работы алгоритма DPLL, реализованного в пакете sat4j, эффективность которого подтверждена итогами конкурса по решению проблемы выполнимости набора высказываний в 2006 г. [5]. Отметим, что этот алгоритм решает задачу вывода только на линейных структурах.

Отметим также, что результаты планирования на линейных задачах превосходят аналогичные результаты, приведенные в [1], т. к. со времени публикации указанной работы скорость работы алгоритма удалось повысить за счет некоторых технических приемов.

На вход машин вывода подавались сгенерированные схемы с возрастающим числом ФС, связанных таким образом, чтобы обеспечить необходимость обработки всех ПВ при осуществлении планирования. Каждой сгенерированной ФС назначалось одинаковое количество аргументов и одинаковое количество целей, что позволяет оценивать скорость относительно количества ФС без учета количества аргументов.

Генерация данных и осуществление вывода проводилось многократно для сглаживания побочных эффектов. При генерации схем с вариантной частью, при участии в схеме n ФС, в каждую вариантную часть помещалось $(n-6)/2$ условных ФС, чтобы обеспечить близкие к максимальным суммарные затраты на обработку условных конструкций согласно формуле (*). Генерация схем с условиями и подпрограммами производилась с таким расчетом, чтобы обеспечить максимальное количество подсхем с максимальной степенью их вложенности. Каждая пара ФС была представлена в виде схемы, включающей в себя вызов других подсхем.

Как видно из представленной на рис. 4 диаграммы, дополнительные затраты на обработку условных ФС замедлили скорость выполнения алгоритма примерно на 1/4 относительно линейного случая. Затраты на обработку подсхем увеличили время работы алгоритма примерно в два раза. Тем

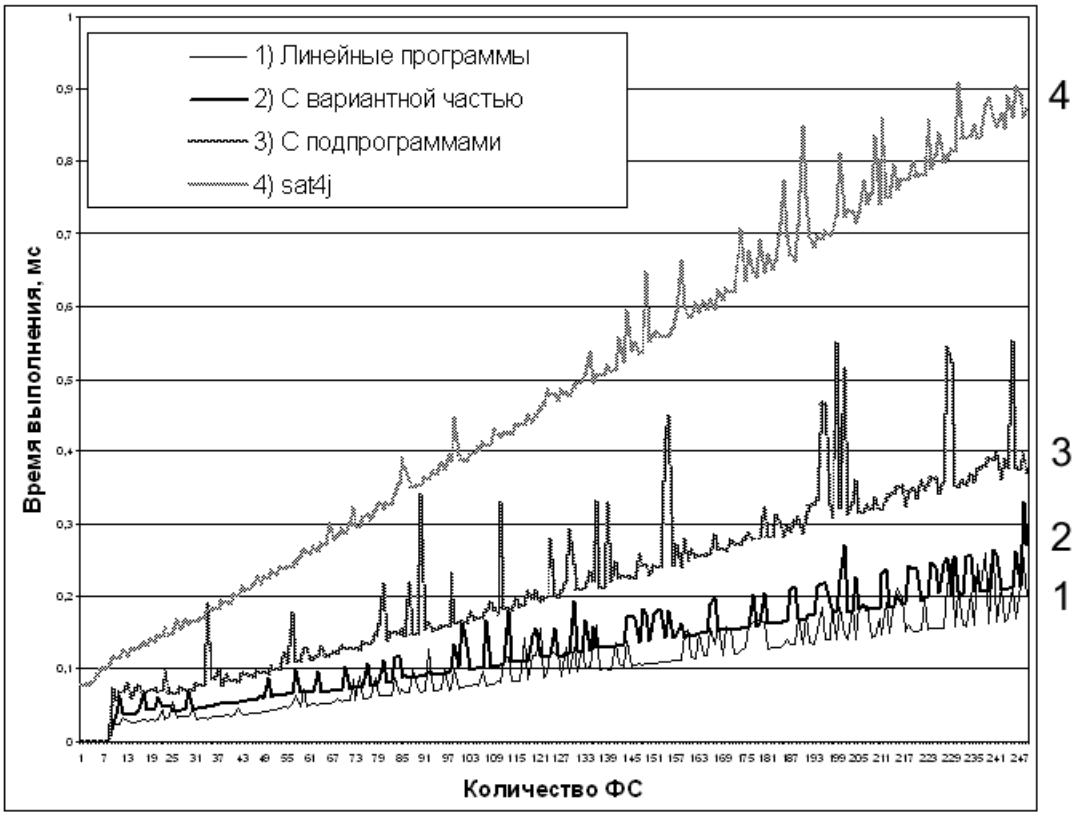

Рис. 4. *Оценка эффективности алгоритма*

не менее, скорость работы предложенного алгоритма даже в самом сложном случае более чем в два раза превышает показатели sat4j. График также достаточно наглядно подтверждает, что алгоритм сохраняет линейные показатели эффективности при обработке схем с условиями и подсхемами.

Заключение

Предложена теоретическая база и реализация алгоритма, предназначенного для синтеза ветвящихся программ и программ с подпрограммами. Экспериментально показано, что предложенная стратегия вывода на базе С-моделей сохраняет эффективность и линейные показатели сложности при планировании на схемах с вариантной частью и подсхемами. Использование стратегии динамической развертки не ухудшает показателей скорости вычислений, но, очевидно, использует значительно меньший объем вычислительных ресурсов при компиляции. В последующих работах будут приведены результаты исследований в области синтеза рекурсивных программ.

СПИСОК ЛИТЕРАТУРЫ

1. Новосельцев В.Б., Пинжин А.Е. Реализация эффективного алгоритма синтеза линейных функциональных программ // Известия Томского политехнического университета. – 2008. – Т. 312. – № 5. – С. 32–35.
2. Новосельцев В.Б. Синтез рекурсивных программ в системах управления пакетами прикладных программ: Дис. ... канд. техн. наук. – Институт теоретической астрономии академии наук СССР. – Л., 1985. – 50 с.
3. Коваленко Д.А., Новосельцев В.Б. Стратегия установления выводимости формул в структурных функциональных моделях // Известия Томского политехнического университета. – 2006. – Т. 309. – № 7. – С. 126–130.
4. Новосельцев В.Б. Теория структурных функциональных моделей // Сибирский математический журнал. – 2006. – Т. 47. – № 5. – С. 1014–1030.
5. SAT-Race 2006: Runtime comparison of all SAT-Race solvers [Электронный ресурс]. – 2006. – Режим доступа: http://fmv.jku.at/sat-race-2006/analysis.html.

Поступила 03.09.2008 г.

УДК 004.89

АЛГОРИТМ СИНТЕЗА ПРОГРАММ С ЯВНОЙ И НЕЯВНОЙ РЕКУРСИЕЙ

А.Е. Пинжин

Томский политехнический университет
E-mail: alex_pinjin@tpu.ru

Теория С-моделей расширяется понятием рекурсивных подсхем. Предлагается алгоритм синтеза рекурсивных программ, где затраты на вывод характеризуются полиномиальной функцией третьей степени. Приведены теоретические и экспериментальные результаты оценки эффективности алгоритма.

Ключевые слова:
Функциональная связь, алгоритмы, логический вывод, синтез программ, функциональное программирование, подпрограммы, условия, рекурсия.

Ранее, в [1] был предложен подход к синтезу программ, содержащих условия и подпрограммы. Предполагалось, что схема не может включать в себя атрибуты сложного типа, объявленные на этой же схеме. Такая ситуация при выполнении доказательства существования решения приводит к бесконечному повторению вывода на одной и той же схеме (зацикливание). В настоящей статье рассматриваются правила введения рекурсивных конструкций в описание схем и предлагаются алгоритмы вывода на рекурсивных подсхемах без использования статической развертки. При этом учитывается случай неявной рекурсии, когда рекурсивный вызов достигается посредством одной или нескольких промежуточных подсхем. Приводятся экспериментальные оценки эффективности предложенного алгоритма.

Правила описания рекурсивных схем

Напомним общий синтаксис описания схемы [2, 3]:

T=(a_0:S_{00}, a_1:S_{01}, ..., a_n:S_{0n}

if $p_1(...)\supset a_{10}$:S_{10}, a_{11}:S_{11}...| $f\_set_1\square...\square p_k(...)\supset a_{k0}$:$S_{k0}$, a_{k1}:S_{k1}... *endif*|f_set_k.

|f_set).

Введение рекурсивных выражений порождает в описаниях схем конструкции следующего вида [4]: T_1=(..., t_2:T_2, ...), T_2=(..., t_3:T_3, ...), ... , T_k=(..., t_1:T_1, ...), где T_1, T_2, T_3, ..., T_k – схемы С-модели М. Будем называть рекурсию *явной*, если k=1, т. е. T_1=(..., t_1:T_1, ...), и *неявной* или *опосредованной*, если k>1. Соответственно, вхождение атрибута t_1:T_1 в схему T_k при k=1, будем называть *явным*, а при k>1 *неявным* или *опосредованным вхождением рекурсивной подсхемы*.